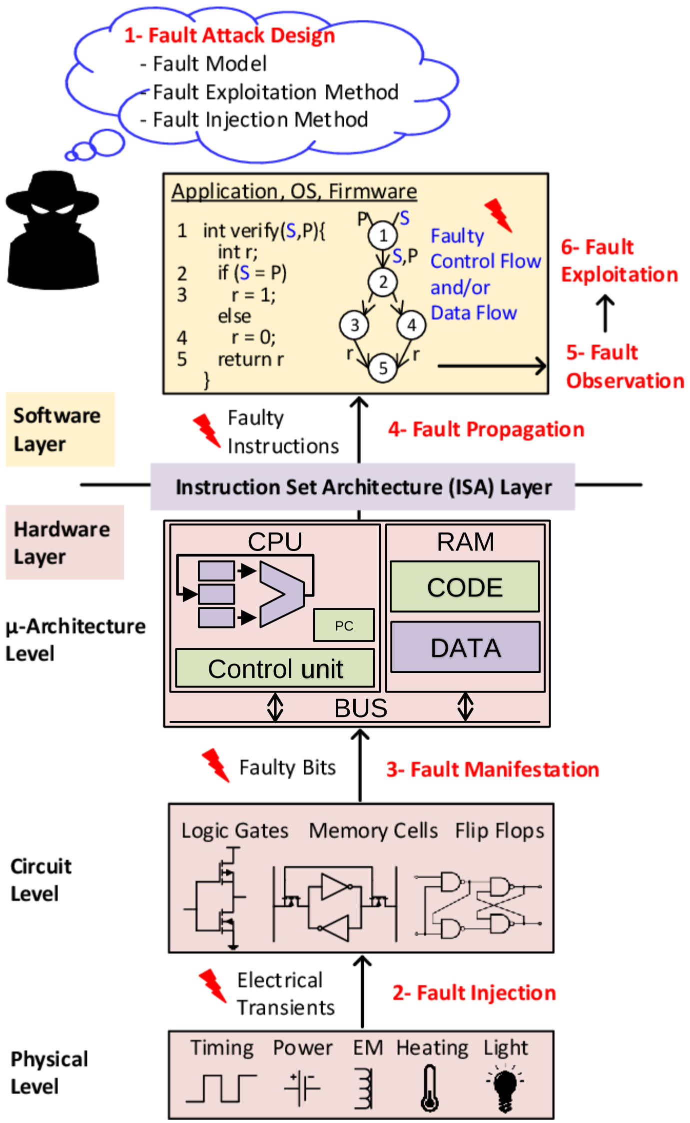
SCI-FI – CONTROL SIGNAL, CODE, AND CONTROL-FLOW INTEGRITY AGAINST FAULT INJECTION ATTACKS

Thomas Chamelot, CEA LIST
Damien Couroussé, CEA LIST
Karine Heydemann, LIP6

CONTEXT

- Fault injection in general purpose processor
  - Extract confidential data
  - Leverage software vulnerabilities
  - Privilege escalation

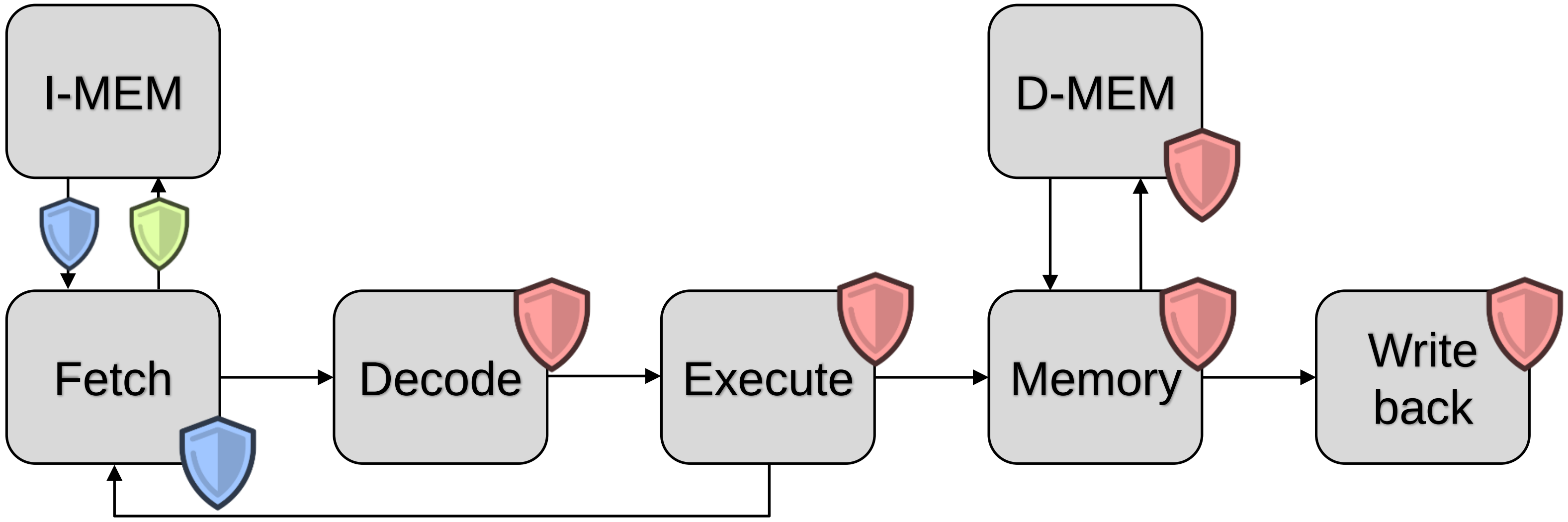


Fault injection attack step by step[1]

[1] Yuce, B. et al.,Fault Attacks on Secure Embedded Software: Threats, Design, and Evaluation. *J Hardw Syst Secure,* 2018

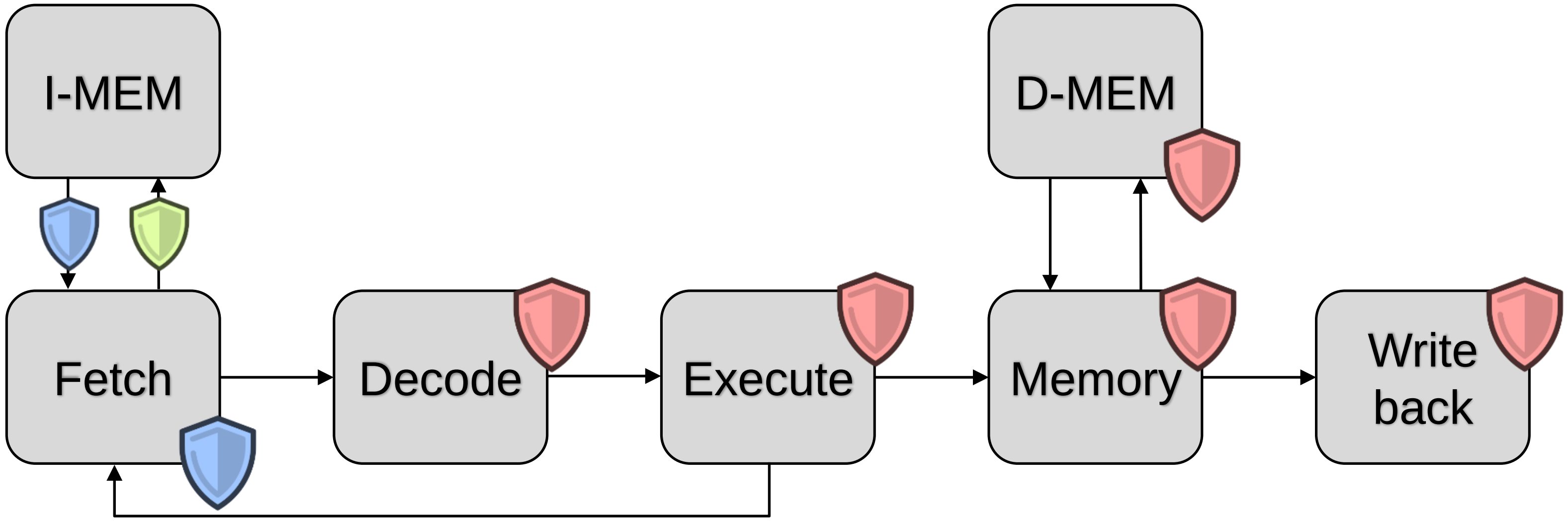
PROTECTIONS AGAINST FAULT INJECTION

- Need 3 security properties
  - Data integrity
  - Control-flow integrity
  - Code integrity



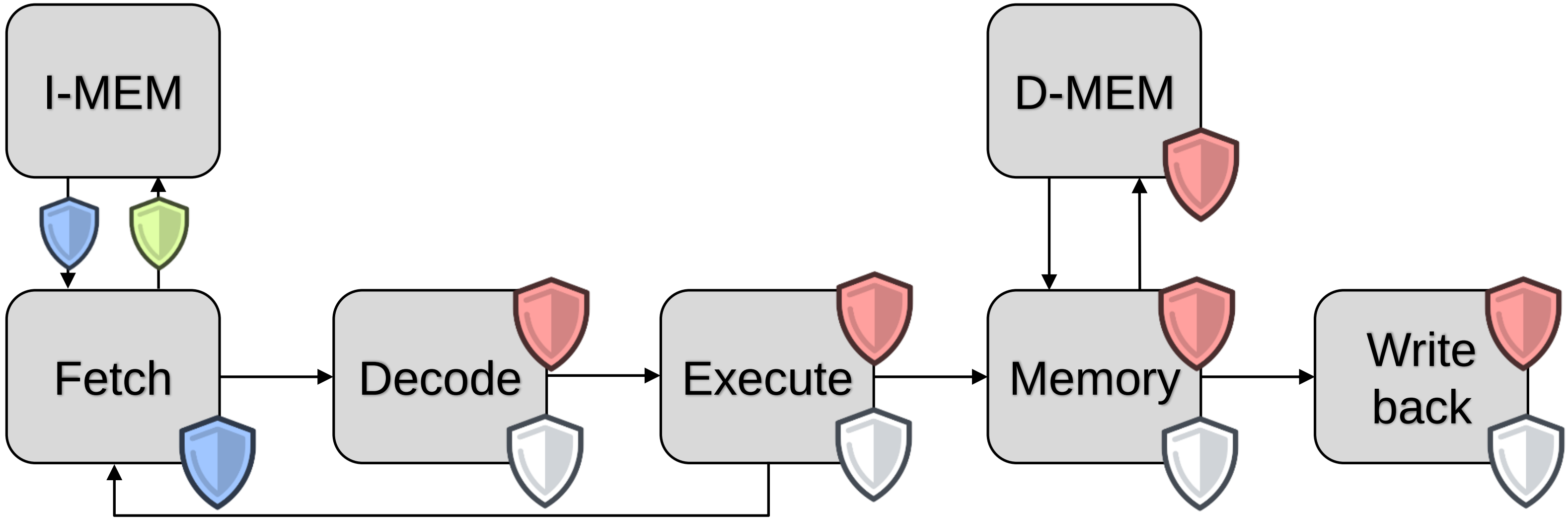
PROTECTIONS AGAINST FAULT INJECTION

- Need 3 security properties
  - Data integrity
  - Control-flow integrity
  - Code integrity
- Requirements
  - Hardware support
  - Program metadata (SW)



PROBLEM

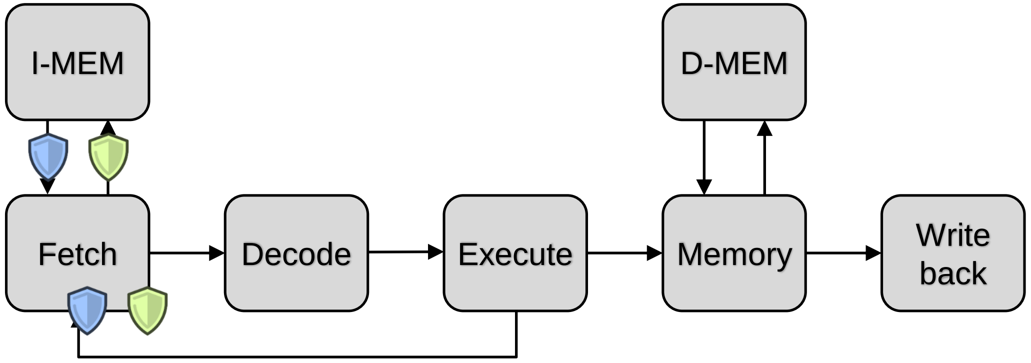
- Need 3 security properties
  - Data integrity
  - Control-flow integrity
  - Code integrity
- Faults in the microarchitecture (Laurent et al [1])
- Need additional property to protect the microarchitecture
  - Execution integrity

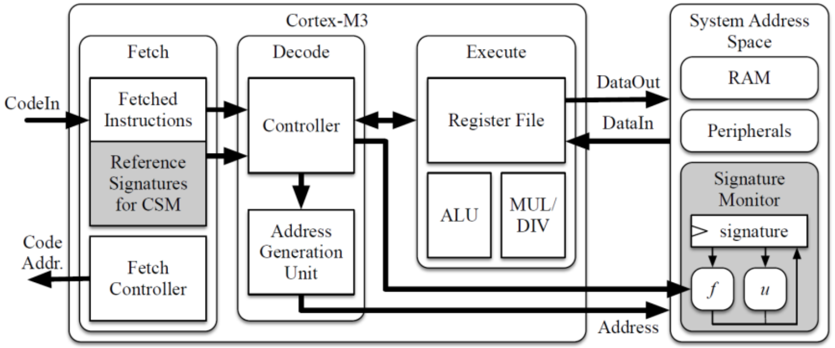


[1] Laurent J. et al., "Cross-layer analysis of software fault models and countermeasures against hardware fault attacks in a RISC-V processor," Microprocessors and Microsystems 2019

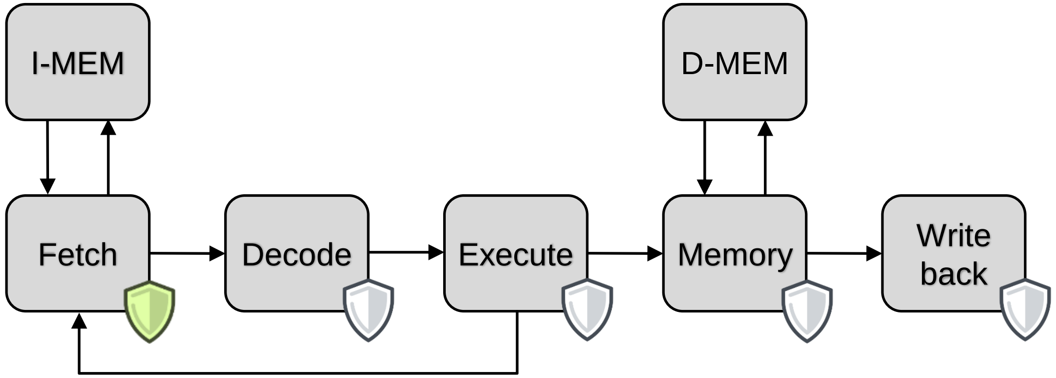
RELATED WORKS

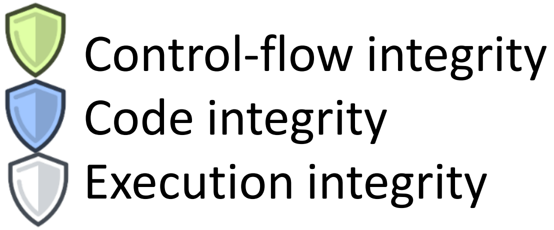
- Protecting the Control Flow of Embedded Processors against Fault Attacks[1]





- On-Line Integrity Monitoring of Microprocessor Control Logic[2]

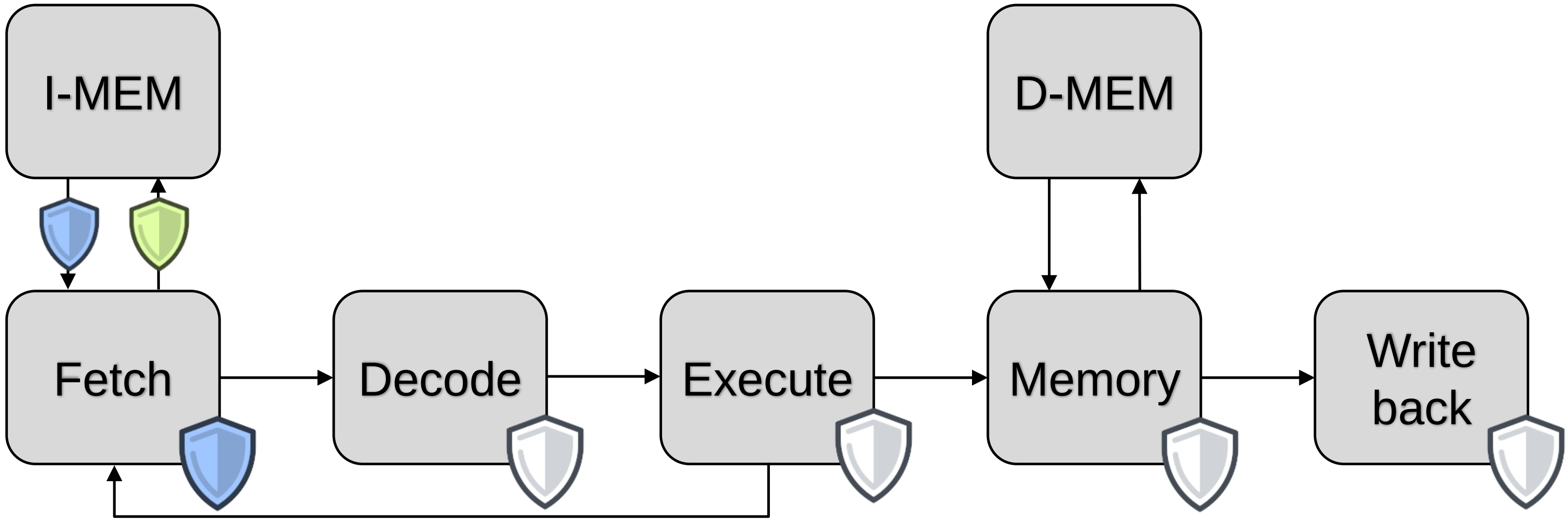




[1] Werner, Mario et al., "Protecting the Control Flow of Embedded Processors against Fault Attacks". In CARDIS 2015
[2] Kim, Seongwoo, et Arun K Somani. "On-Line Integrity Monitoring of Microprocessor Control Logic ". Microelectronics Journal, 2001.

GOALS & CHALLENGES

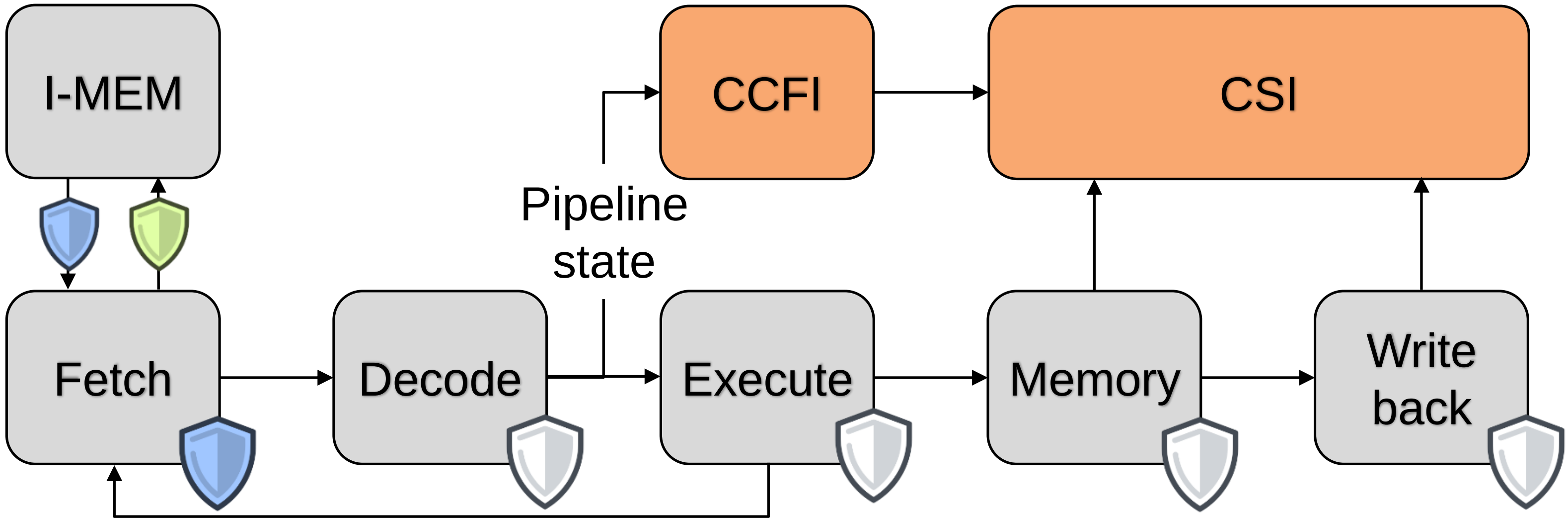
- Goals
  - Support simultaneously code, control-flow and execution integrity
  - Execution integrity as processor’s control signal integrity
- Challenges
  - Design an efficient mechanism for execution integrity
  - Combine execution integrity with code and control-flow integrity



Control-flow integrity
Code integrity
Execution integrity

PROPOSAL

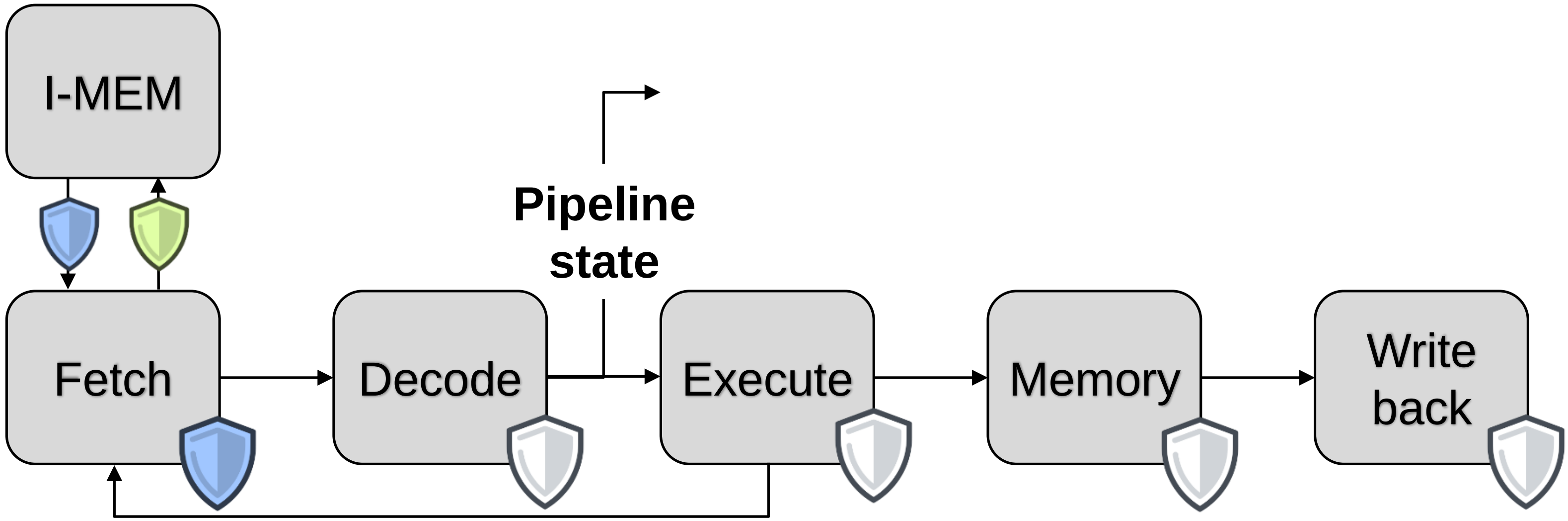
- SCI-FI – Control Signal, Code, and Control Flow Integrity Against Fault Injection
  - CCFI: Signature-based mechanism for the pipeline frontend
    - Provides code, control-flow and execution integrity
    - Needs compiler and static analysis support to compute reference signatures
  - CSI: Redundancy-based mechanism for the pipeline backend
    - Provides execution integrity



Control-flow integrity
Code integrity
Execution integrity

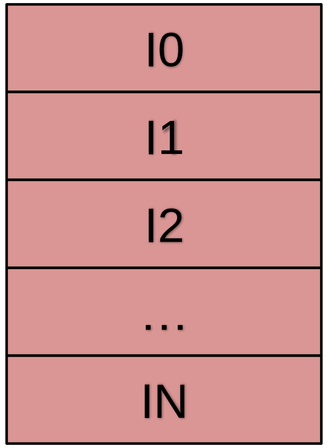
PIPELINE STATE

- Control signals outputted by the decode stage and fed to CCFI
  - Computable by static analysis (for reference signatures)
  - Static control signals: depend on the instruction only
    - Operands selection
    - Operation control (ALU, LSU)
    - Immediate
  - Dynamic control signals: depend on instruction sequence but not on data
    - Forwarding mechanism



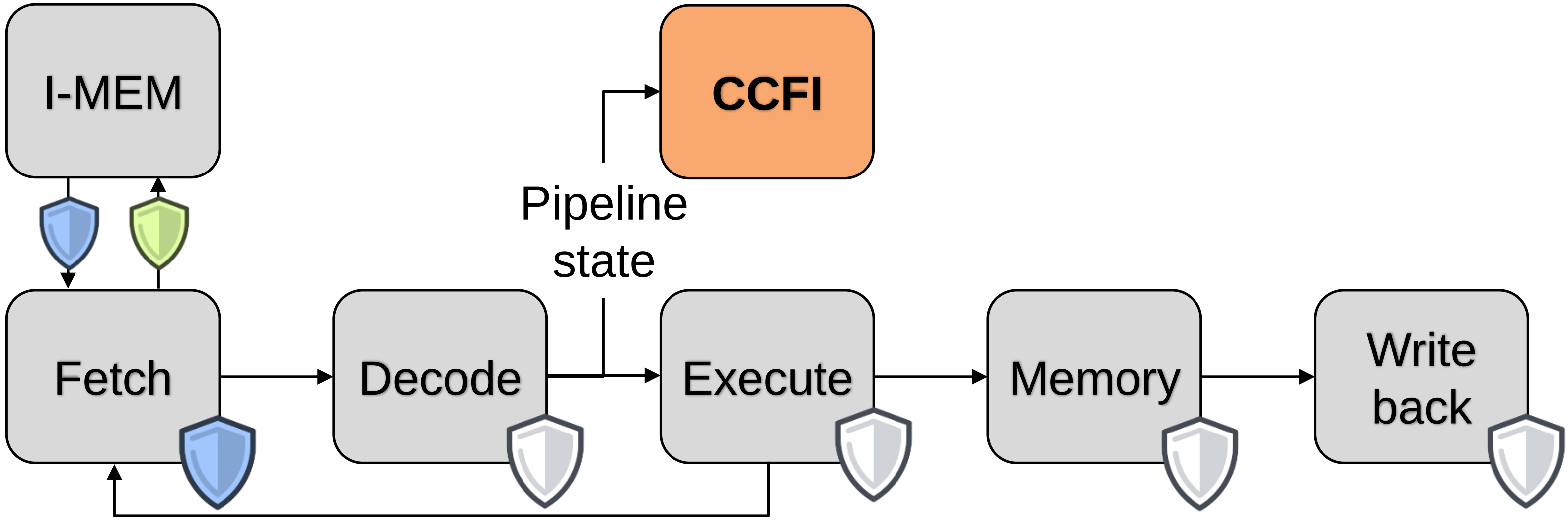
Control-flow integrity
Code integrity
Execution integrity

CCFI : SIGNATURE FUNCTION



$S_0 = f(\Sigma_0, IV)$

- Σ_i pipeline state associated to instruction I
- $S_i = f(\Sigma_i, IV)$
- Properties to guarantee CI, CFI, EI
  - Collision resistance $P\left[f(\Sigma_i, IV) \neq f(\Sigma_j, IV)\right] < \varepsilon, \forall\, \Sigma_i \neq \Sigma_j$
  - Error preservation $f(\Sigma_i \oplus \Delta_i, IV) = S_i \oplus \delta_i, \forall \Delta_i \neq 0 \rightarrow \delta_i \neq 0$
  - Non associativity $f(\Sigma_i, f(\Sigma_j, IV)) \neq f\left(\Sigma_j, f(\Sigma_i, IV)\right), \forall \Sigma_i \neq \Sigma_j$
- Constraints
  - Execute in 1 cycle
  - Small hardware area



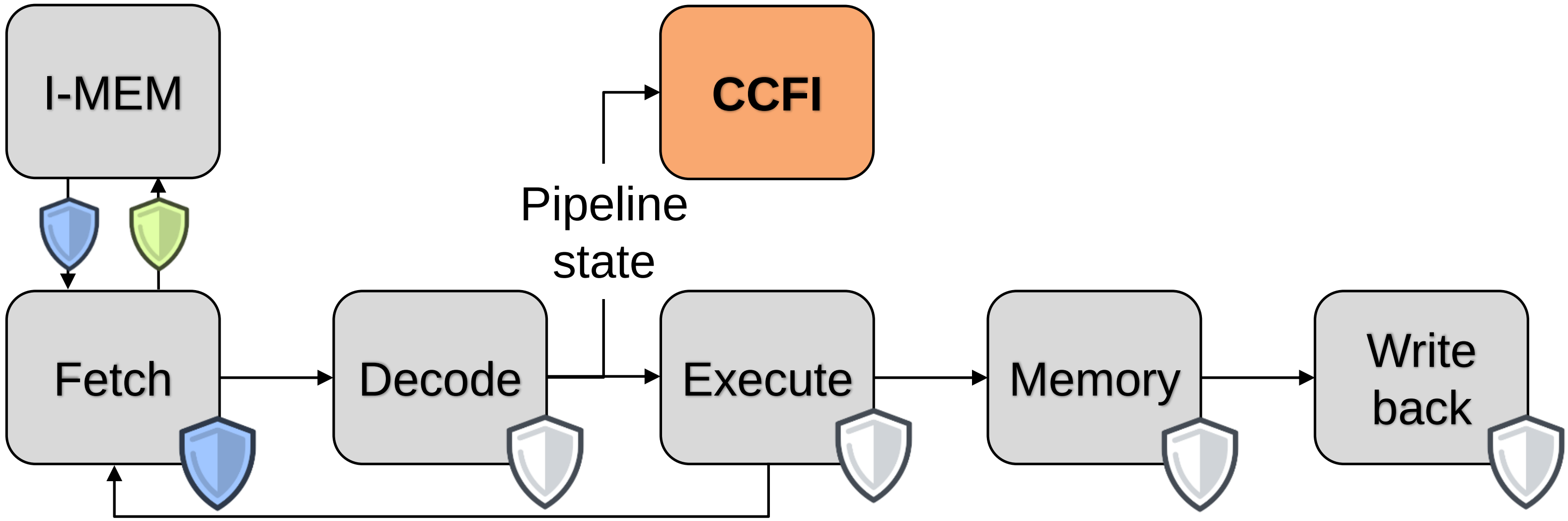
Control-flow integrity
Code integrity
Execution integrity

CCFI : SIGNATURE FUNCTION

I0
I1
I2
…
IN

$S_0 = f(\Sigma_0, IV)$
$S_1 = f(\Sigma_1, S_0)$

- Σ_i pipeline state associated to instruction I
- $S_i = f(\Sigma_i, IV)$
- Properties to guarantee CI, CFI, EI
  - Collision resistance $P\left[f(\Sigma_i, IV) \neq f\left(\Sigma_j, IV\right)\right] < \varepsilon, \forall\, \Sigma_i \neq \Sigma_j$
  - Error preservation $f(\Sigma_i \oplus \Delta_i, IV) = S_i \oplus \delta_i, \forall \Delta_i \neq 0 \rightarrow \delta_i \neq 0$
  - Non associativity $f\left(\Sigma_i, f(\Sigma_j, IV)\right) \neq f\left(\Sigma_j, f(\Sigma_i, IV)\right), \forall \Sigma_i \neq \Sigma_j$
- Constraints
  - Execute in 1 cycle
  - Small hardware area

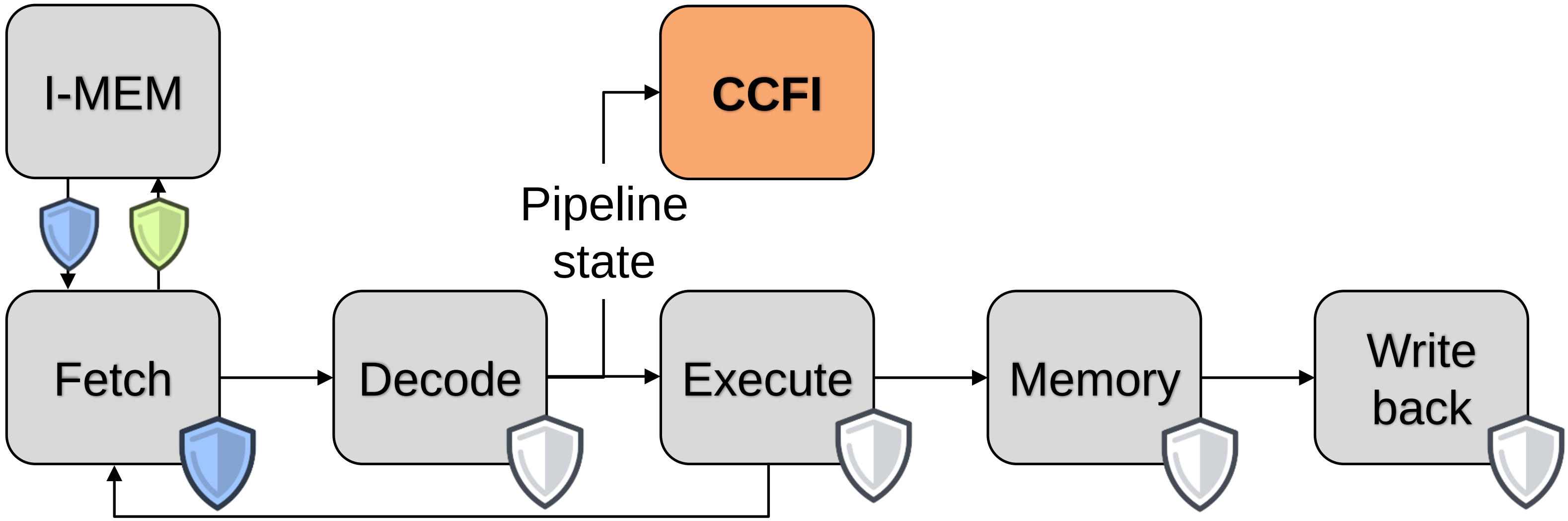


Control-flow integrity
Code integrity
Execution integrity

CCFI : SIGNATURE FUNCTION

- Σ_i pipeline state associated to instruction I
- $S_i = f(\Sigma_i, IV)$
- Properties to guarantee CI, CFI, EI
  - Collision resistance $P[f(\Sigma_i, IV) \neq f(\Sigma_j, IV)] < \varepsilon, \forall\, \Sigma_i \neq \Sigma_j$
  - Error preservation $f(\Sigma_i \oplus \Delta_i, IV) = S_i \oplus \delta_i, \forall \Delta_i \neq 0 \rightarrow \delta_i \neq 0$
  - Non associativity $f(\Sigma_i, f(\Sigma_j, IV)) \neq f\left(\Sigma_j, f(\Sigma_i, IV)\right), \forall \Sigma_i \neq \Sigma_j$
- Constraints
  - Execute in 1 cycle
  - Small hardware area

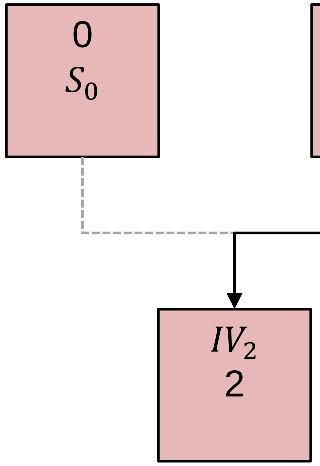
I0	$S_0 = f(\Sigma_0, IV)$
I1	$S_1 = f(\Sigma_1, S_0)$
I2	$S_2 = f(\Sigma_2, S_1)$
…	
IN	$S_N = f(\Sigma_N, S_{N-1})$



Control-flow integrity
Code integrity
Execution integrity

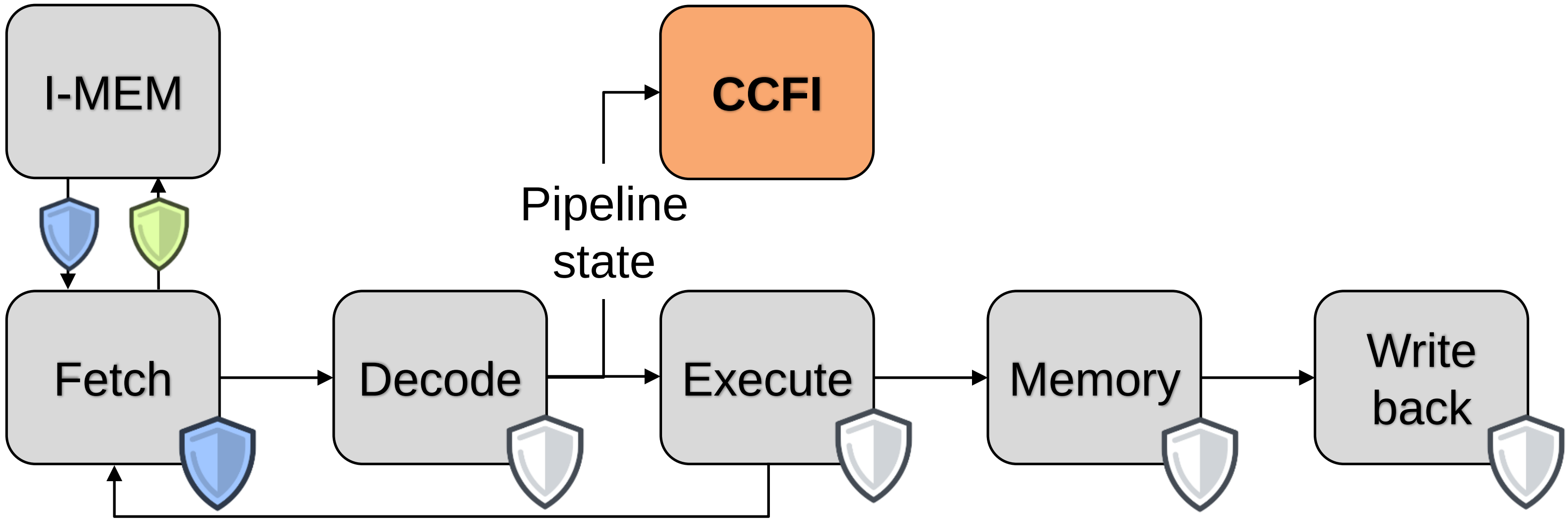
CCFI : MERGING EXECUTION PATHS

- Problem
  - N predecessors => N IV
  - N IV => N signatures
  - CCFI requires a unique IV per basic block
- Solution
  - Update mechanism



Issue:

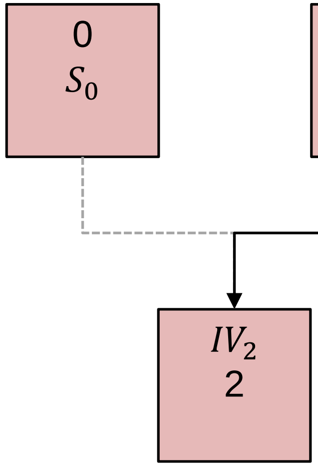
$$IV_2 = \begin{cases} S_0 \; if \; pred = BB_0 \\ S_1 \; if \; pred = BB_1 \end{cases}$$



Control-flow integrity
Code integrity
Execution integrity

CCFI : UPDATE MECHANISM

- $S' = u(S,\ P)$: update S using patch value P
- Properties of u for CI, CFI, EI
  - Surjection $S' = u(S, P),\ \forall S', \forall S,\ \exists P$
  - Error preservation $u(S \oplus \Delta_i, P) = S' \oplus \delta_i, \forall \Delta_i \neq 0 \rightarrow \delta_i \neq 0$
  - Invertibility $P = u^{-1}(S, S'), \forall S, \forall S'$



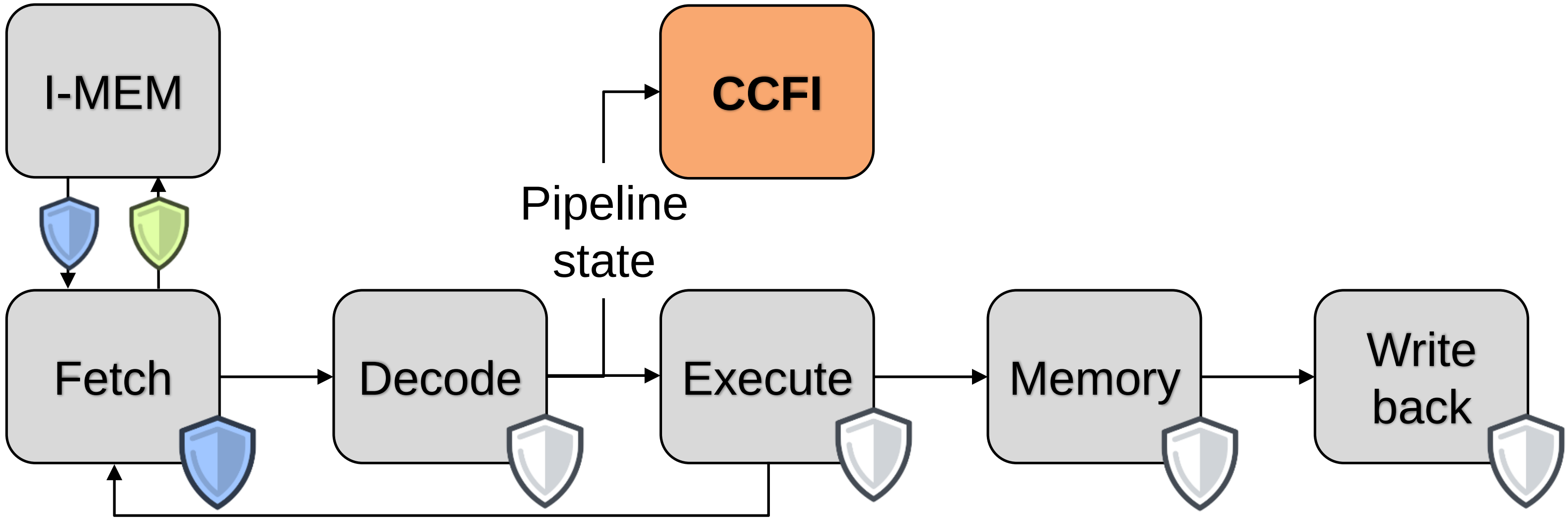
Solution:

$$IV_2 = \begin{cases} S_0 \ if\ pred = BB_0 \\ u(S_1, P)\ if\ pred = BB_1 \end{cases}$$

$$u(S_1, P) = S_0$$

- Patch loaded in dedicated register by custom instruction
- Patch reset to P_0 after branch

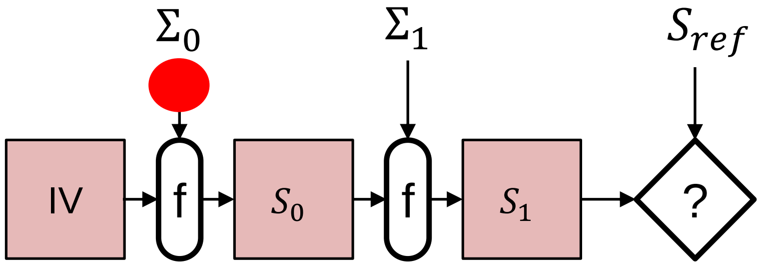
$$S = u(S,\ P_0)$$



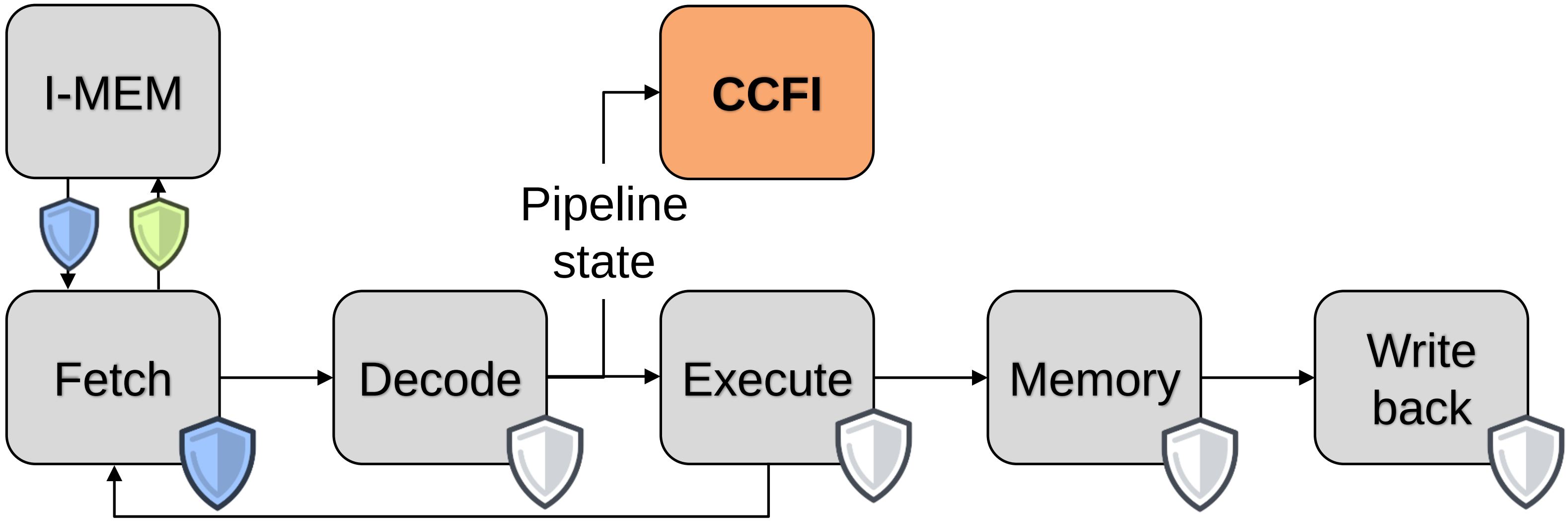
- Limitation
  - No indirect branches

Control-flow integrity
Code integrity
Execution integrity

CCFI : SIGNATURE VERIFICATION

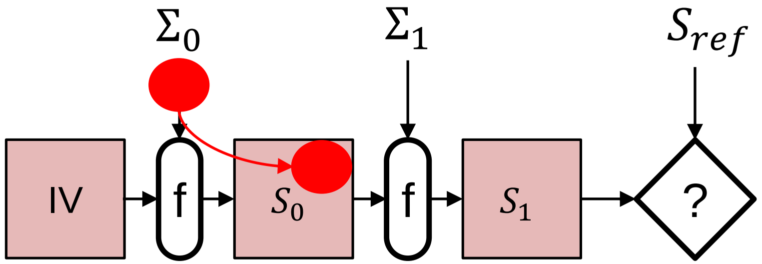


- 1 signature for each instruction
- Any captured fault is forwarded
- Can be placed anywhere
- Verification supported by dedicated control-flow instructions
  - Load reference signature located just after in memory
  - Trigger verification
  - Behave as standard control-flow instructions

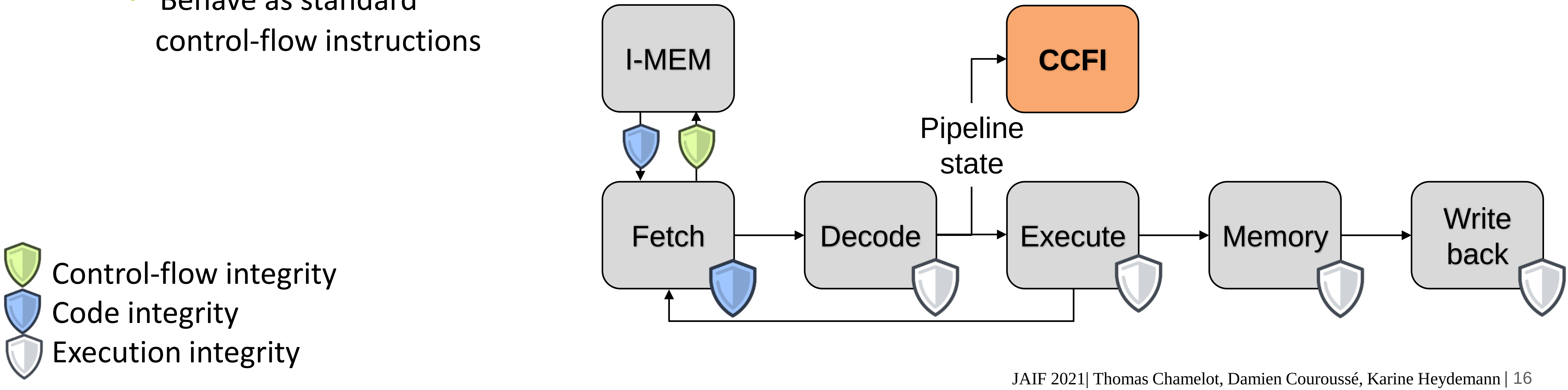


Control-flow integrity
Code integrity
Execution integrity

CCFI : SIGNATURE VERIFICATION

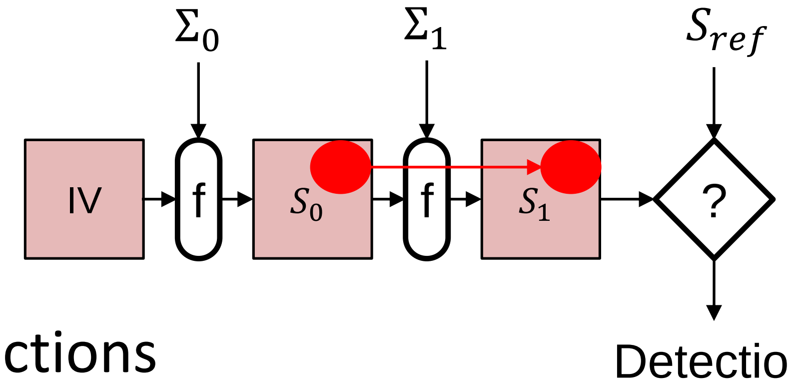


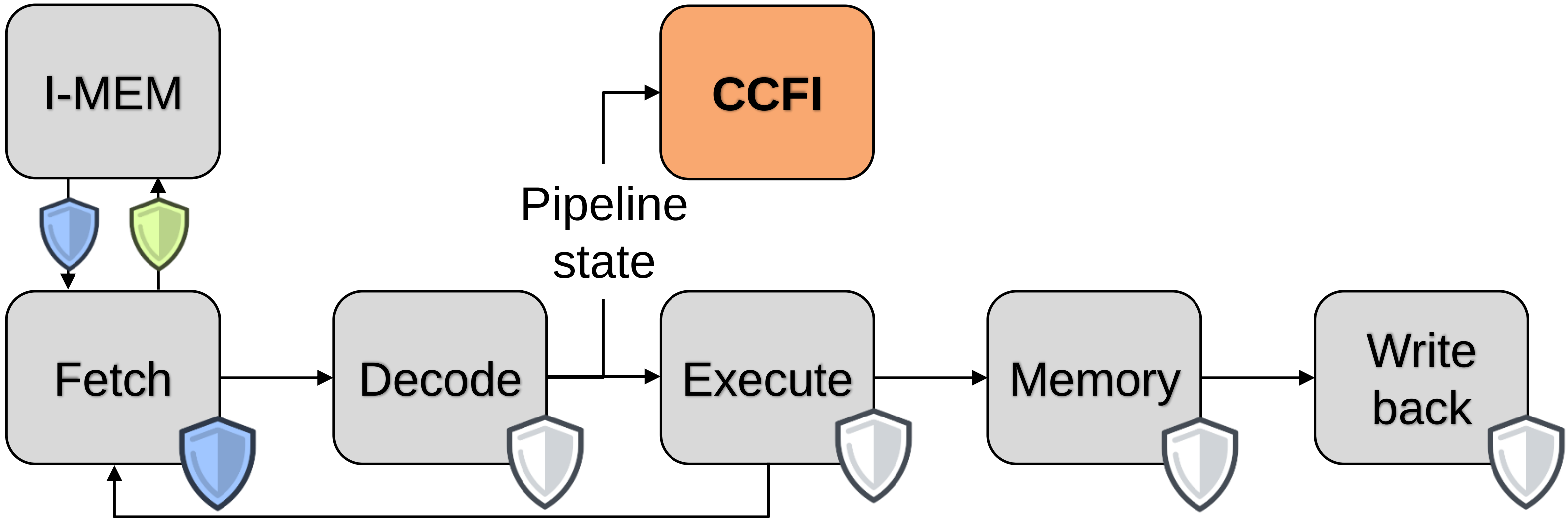
- 1 signature for each instruction
- Any captured fault is forwarded
- Can be placed anywhere
- Verification supported by dedicated control-flow instructions
  - Load reference signature located just after in memory
  - Trigger verification
  - Behave as standard control-flow instructions



CCFI : SIGNATURE VERIFICATION

- 1 signature for each instruction
- Any captured fault is forwarded
- Can be placed anywhere
- Verification supported by dedicated control-flow instructions
  - Load reference signature located just after in memory
  - Trigger verification
  - Behave as standard control-flow instructions

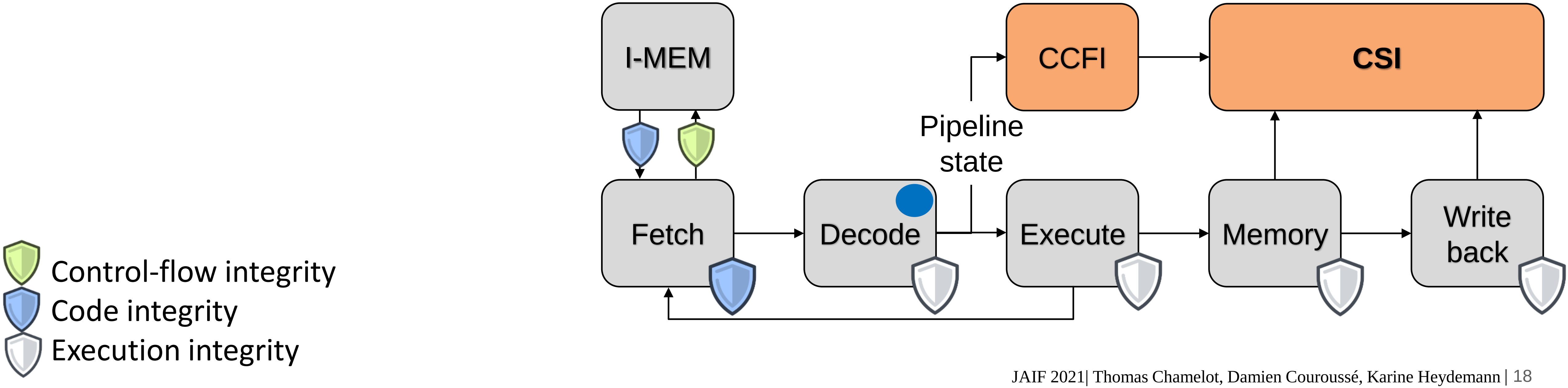




Control-flow integrity
Code integrity
Execution integrity

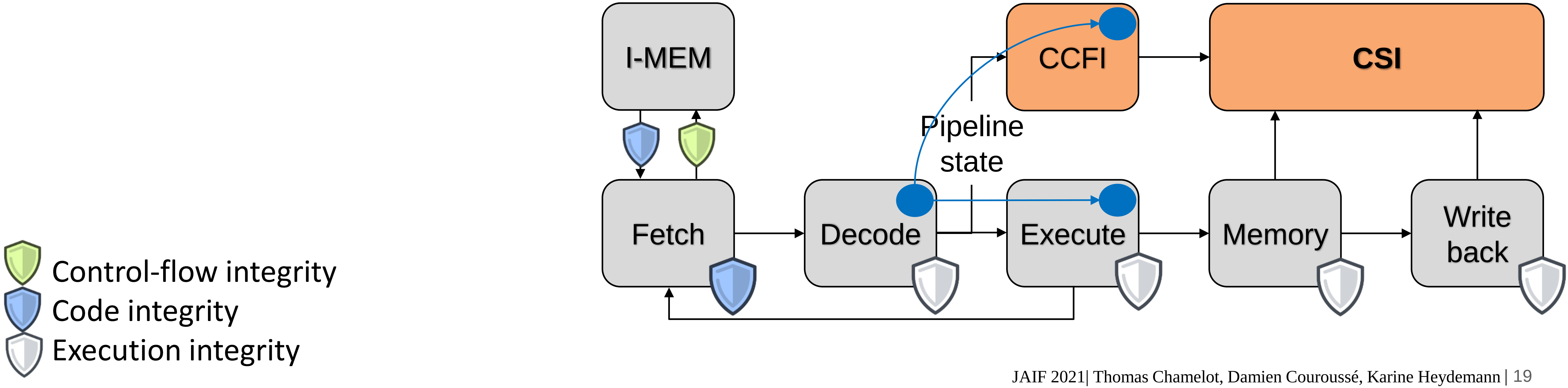
CSI – CONTROL SIGNALS INTEGRITY

- Duplicates signals from the pipeline stage
- Checks original against its duplicate between each stage
- Can use different redundancy scheme
  - Simple copy
  - Complementary copy
  - XOR with constant



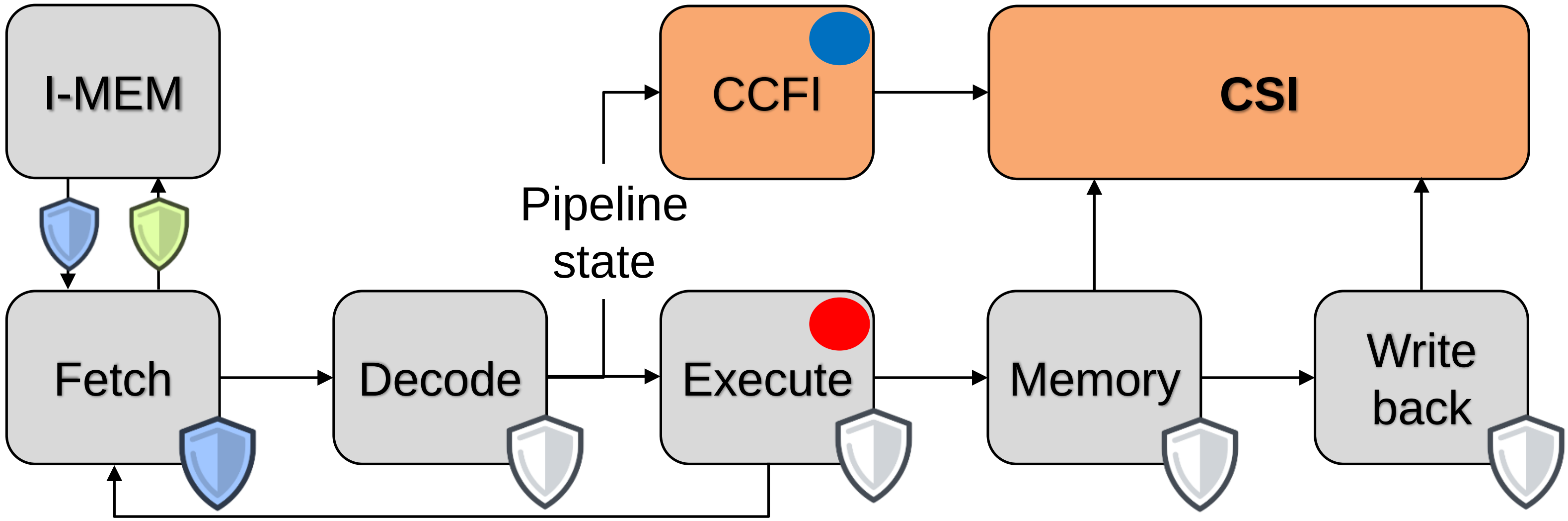
CSI – CONTROL SIGNALS INTEGRITY

- Duplicates signals from the pipeline stage
- Checks original against its duplicate between each stage
- Can use different redundancy scheme
  - Simple copy
  - Complementary copy
  - XOR with constant



CSI – CONTROL SIGNALS INTEGRITY

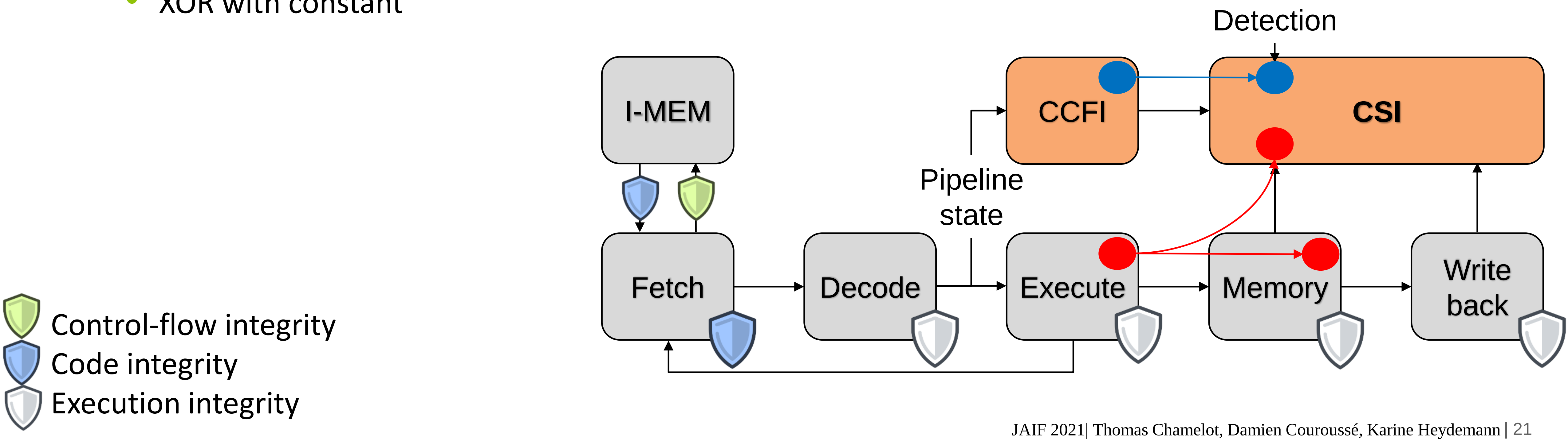
- Duplicates signals from the pipeline stage
- Checks original against its duplicate between each stage
- Can use different redundancy scheme
  - Simple copy
  - Complementary copy
  - XOR with constant



Control-flow integrity
Code integrity
Execution integrity

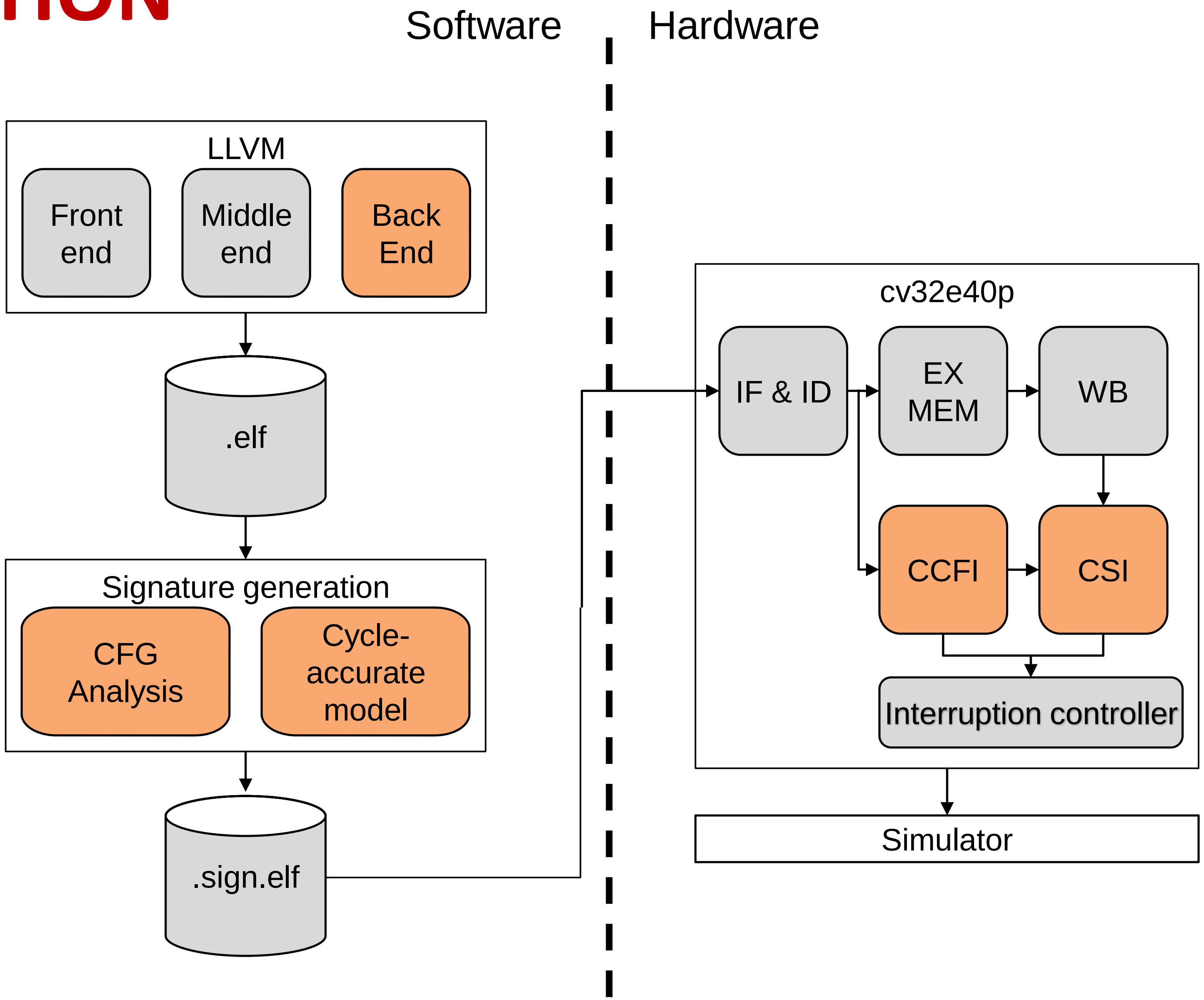
CSI – CONTROL SIGNALS INTEGRITY

- Duplicates signals from the pipeline stage
- Checks original against its duplicate between each stage
- Can use different redundancy scheme
  - Simple copy
  - Complementary copy
  - XOR with constant



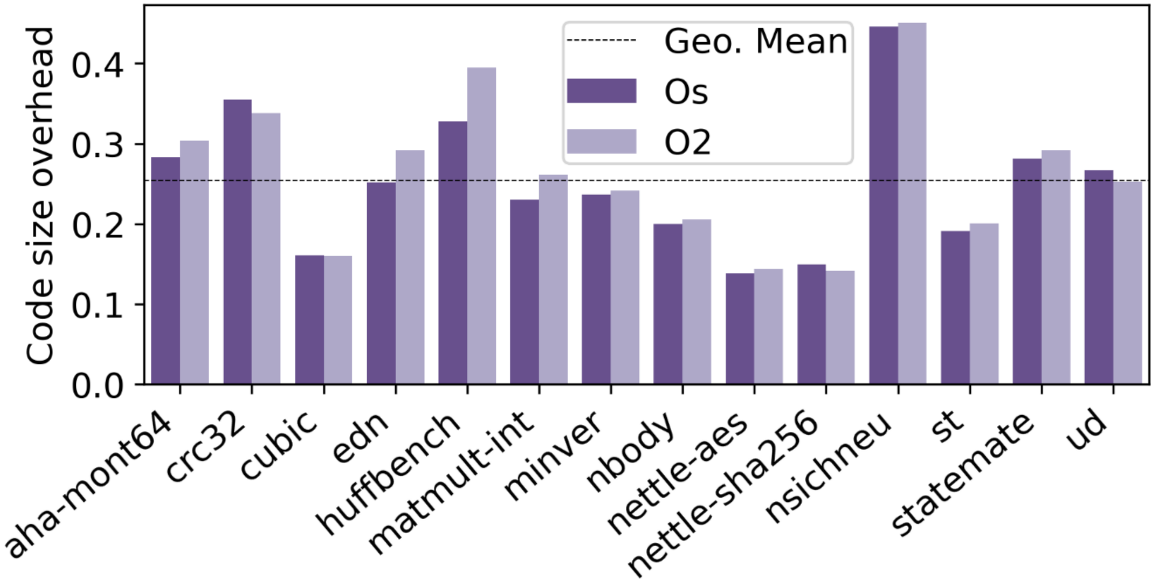
IMPLEMENTATION

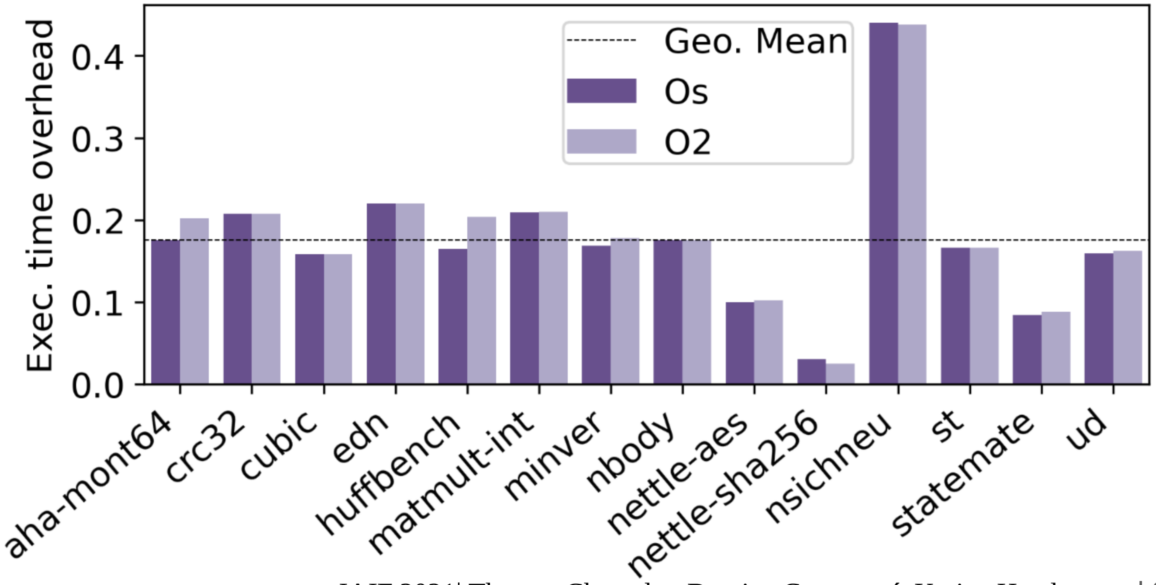
- Processor: CV32E40P
  - ISA: RV32IMC
  - Pipeline: 4-stages, in-order
- Signature function
  - CRC32
  - CBC-MAC Prince (code authenticity)
- Update function
  - XOR
- Redundancy scheme
  - Simple copy
- Toolchain
  - LLVM (RISC-V backend) & Newlib
  - Custom signature generation tool



EXPERIMENTAL EVALUATIONS

- Hardware overhead (ASIC 22nm FDSOI @ 400MHz)
  - CRC32: 6.5%, 55kGE (+5kGE)
  - Prince: 23.8%, 64kGE (+13kGE)
- Software overhead (Embench-IOT, cycle accurate HDL simulation)
  - Average code size: 25.4%, [13.8, 45.1]%
  - Average execution time: 17.5%, [2.5, 44]%





CONCLUSION

- SCI-FI: a new counter-measure for **Code, Control-Flow and Execution Integrity**
  - Signature-based mechanism for the pipeline frontend
  - Redundancy-based mechanism for the pipeline backend
  - Architecture is highly flexible: additional code authenticity
  - Full software stack and hardware support
  - Low hardware overhead regarding complete system with memory (+13kGE)
- Future work
  - Support for indirect branches
  - Combination with authenticated decryption protection